

AD-A105 028

MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB
PARALLEL STRING PARSING USING LATTICE GRAPHS.(U)
MAY 81 A ROSENFELD

F/G 9/2

AFOSR-77-3271

UNCLASSIFIED

TR-1036

AFOSR-TR-81-0623

NL

1 of 1
AD-A
1050-2



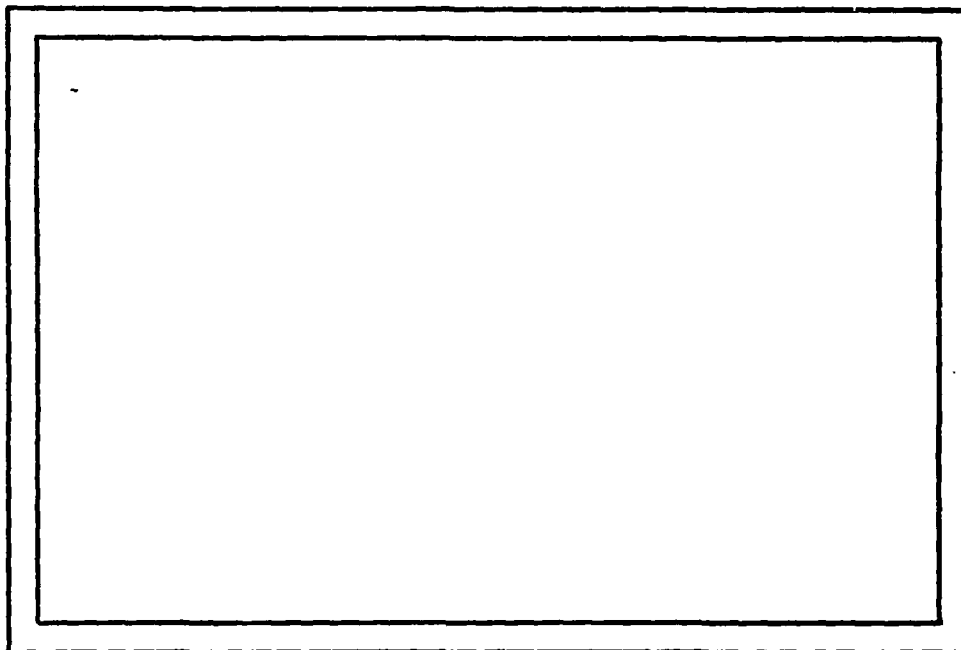
END
DATE
FILMED
10-81
DTIC

AFOSR-TR. 81-0623

LEVEL 11

10

AD A105028



COMPUTER SCIENCE
TECHNICAL REPORT SERIES



DTIC
ELECTE
OCT 6 1981
A

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND

20742

FILE COPY

0 5 006

Approved for public release;
distribution unlimited.

TR-1036
AFOSR-77-3271

May 1981

PARALLEL STRING PARSING
USING LATTICE GRAPHS

Azriel Rosenfeld

(Computer Vision Laboratory)
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

Given a grammar G and a string σ , all possible parses of σ can be constructed by repeatedly applying the rules of G in parallel. This process creates a "lattice graph" in which any directed path from the least element to the greatest element is a sentential form that occurs in a (partial) parse of σ . Examples are given illustrating how, at least for some grammars, this process does not lead to a combinatorial explosion, and could thus be used to parse strings very rapidly if suitable parallel hardware were available.

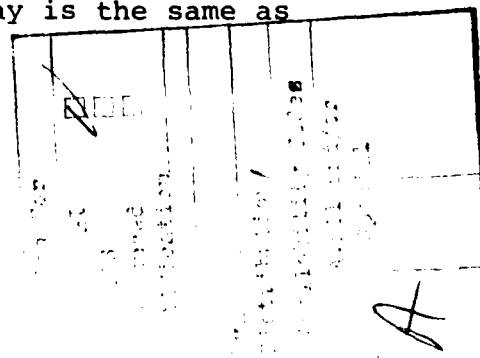
The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Sherry Palmer in preparing this report.

1. Introduction

Let G be a grammar, which we shall first assume for simplicity to be context-free, with rules of the form $A \rightarrow \alpha$ (α non-null). Conventionally, to parse a string σ with respect to G , we find a match in σ to the right-hand side (RHS) of some rule $A \rightarrow \alpha$; replace this instance of α by A ; and repeat the process, until σ is reduced to a single S (the "start symbol" of G).

Time bounds on context-free parsing have been extensively studied (e.g., [1]). For general context-free grammars, the time required to parse a string of length n is on the order of n^3 ; and for non-context-free grammars the situation is presumably worse.

One could image parsing "in parallel" by replacing many α_i 's by A_i 's simultaneously, but this leads to difficulties if the RHSs overlap; for example, if $A \rightarrow \alpha$ and $B \rightarrow \beta$ are rules, where β is a substring of α , where do we put the B relative to the A when we apply both rules? In [2] it is suggested that parallel parsing can be done by a two-step process, first choosing nondeterministically which RHSs should be rewritten, and then actually rewriting them iff no overlapping RHS was chosen for rewriting. It is shown in [2] that the language parsed in this way is the same as the language of G as ordinarily defined.



This note describes an alternative approach in which we actually rewrite all RHSs in parallel, and represent the result not by a string but by a "lattice graph" in which paths between the terminal points correspond to sentential forms. It is easily seen that this process yields all possible parses of the given string. Examples are given illustrating how, at least for some grammars, this process does not lead to a combinatorial explosion. Implementation of this approach using a reconfigurable network of processors is also briefly discussed. Such an implementation would permit parsing to be carried out rapidly, with the time required depending primarily on the height of the parse tree.

2. Parallel parsing

Let G be an acyclic directed graph with set of nodes N . If there is a (directed) path from p to q , where p, q are in N , we say that $p \leq q$. Evidently, $\leq$ is a partial order relation: reflexive ($p \leq p$ for all p), antisymmetric ($p \leq q$ and $q \leq p$ imply $p = q$), and transitive ($p \leq q$ and $q \leq r$ imply $p \leq r$). We say that g is the greatest lower bound (glb) of a set of nodes $N' \subseteq N$ if $g \leq p$ for all $p \in N'$, and $g' \leq p$ for all $p \in N'$ implies $g' \leq g$. Similarly, we say that l is the least upper bound (lub) of N' if $p \leq l$ for all $p \in N'$, and $p \leq l'$ for all $p \in N'$ implies $l \leq l'$. We call G a lattice graph if every nonempty $N' \subseteq N$ has a glb and an lub. In particular, the glb and lub of N itself exist; we denote them by 0 and 1 , respectively. Note that there is a path from 0 to any $p \in N$, and from any $p \in N$ to 1 , so that G is connected. If (p, q) is an arc of G (so that $p \leq q$), we call p a predecessor of q and q a successor of p .

Any string $x_0 \dots x_n$ may be regarded as an acyclic directed graph, with arcs between successive symbols (x_{i-1}, x_i) , $1 \leq i \leq n$. Evidently a string is a lattice graph. We shall assume that the given string σ which is to be parsed begins and ends with endmarkers, say $\$ \sigma \epsilon$; thus $\$ = 0$ and $\epsilon = 1$ when we regard the string as a lattice graph.

Let α be a substring of σ , and let $A \rightarrow \alpha$ be a rule of the grammar G , where $\alpha = x_i \dots x_j$ (say). When we apply this rule to σ , we create a "short cut" through A from the

predecessor x_{i-1} of α to its successor x_{j+1} :

$$\dots x_{i-1} \overbrace{x_i \dots x_j}^A x_{j+1} \dots$$

(Here the bar extends from just after the predecessor of A to just before its successor.) Evidently the result is still a lattice graph, though it is no longer a string.

The situation is analogous if we apply many rules to σ simultaneously, even if their RHSs overlap; we still obtain a lattice graph, e.g.

$$\dots \overbrace{w \dots x \dots y \dots z}^A \dots$$

Note that any directed path from \$ to ϕ through this graph represents a possible sentential form derivable from σ using the grammar G.

After the first round of (parallel) rule application, we no longer have a lattice graph that is a string, but any directed path from \$ to ϕ is a string, and we can still apply rules to the lattice graph, by matching their RHSs against all possible substrings of these strings. Such applications create further "short cuts" in the graph, but it remains a lattice graph. If we ever create a short cut in which we can go directly from \$ to ϕ via a single S (the start symbol of G), we have successfully parsed σ .

To clarify these ideas, we give a simple example. Let G be the parenthesis string grammar whose rules are

$$S \rightarrow SS \quad S \rightarrow (S) \quad S \rightarrow ()$$

and let σ be the string $\$ (() ()) () \phi$

In the first round of parallel parsing, the rule $S \rightarrow ()$ applies in four places, yielding the graph

$$\begin{array}{c} \$ (() ()) () \$ \\ \bar{S}_1 \bar{S}_1 \bar{S}_1 \bar{S}_1 \end{array}$$

where the subscript indicates the round number. Note that none of these rules have overlapping RHSs. In the second round, there are many possible paths from \$ to \$, e.g. $\$(S())S() \$$, but the only ones that allow new rule applications are those that contain two or more consecutive Ss. Thus the second round yields

$$\begin{array}{c} S_2 \\ \$ (() ()) () \$ \\ \bar{S}_1 \bar{S}_1 \bar{S}_1 \bar{S}_1 \\ \hline S_2 \end{array}$$

where the two rules now do have overlapping RHSs. At the third round, we still have new rule applications for the paths that contain two consecutive Ss - i.e., the paths for which the first two or the last two Ss were rewritten, but not all three; but both of those paths yield the same new path:

$$\begin{array}{c} S_2 \\ \$ (() ()) () \$ \\ \bar{S}_1 \bar{S}_1 \bar{S}_1 \bar{S}_1 \\ \hline S_2 \\ \hline S_3 \end{array}$$

The fourth round thus allows us to rewrite (S_3) as S_4 ,
and the fifth round to rewrite S_4S_1 as S_5 , completing
the parse of σ :

$$\begin{array}{c}
 S_2 \\
 \hline
 \$ (() () ()) () \$ \\
 \hline
 \begin{array}{cccc}
 S_1 & S_1 & S_1 & S_1
 \end{array} \\
 \hline
 S_2 \\
 \hline
 S_3 \\
 \hline
 S_4 \\
 \hline
 S_5
 \end{array}$$

In this example there is an ambiguity (the two S_2 's)
in which both alternatives lead to the same result, but there
are no "dead ends" (rule application sequences which do not
lead to a parse). As another example, consider the grammar
for palindromes of even length whose rules are

$$S \rightarrow aa \quad S \rightarrow bb \quad S \rightarrow aSa \quad S \rightarrow bSb$$

and let σ be the string

$$\$aaabbbaaa\$$$

Here at the first round we have

$$\begin{array}{c}
 S_1 \quad S_1 \\
 \hline
 \$aaabbbaaa\$ \\
 \hline
 S_1 \quad S_1 \quad S_1
 \end{array}$$

where only the center S is part of a correct parse. In fact,
at the next round the other four S s do not contribute to rule
applications; only the center one gives us

$$\begin{array}{c}
 S_1 \quad S_1 \\
 \hline
 \$aaabbbaaa\$ \\
 \hline
 S_1 \quad S_1 \quad S_1 \\
 \hline
 S_2
 \end{array}$$

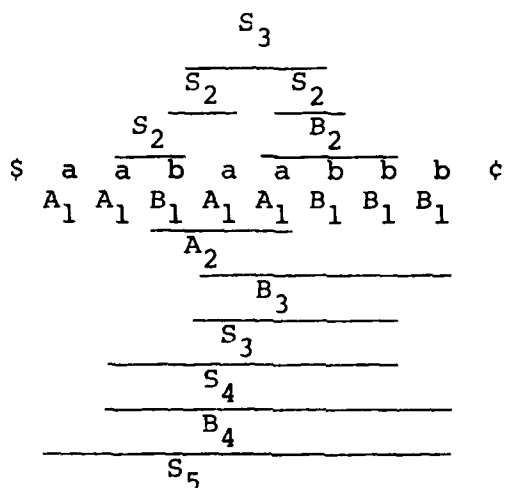
and similarly at successive rounds we rewrite aS_2a as S_3 and
 aS_3a as S_4 , completing the parse of σ .

A third example uses the following grammar for the set of strings that contain equal numbers of a's and b's:

$S \rightarrow aBS$ or bAS or aB or bA

$A \rightarrow bAA$ or a ; $B \rightarrow aBB$ or b

For the string aabaabbb we have the following parallel parse:



The reduction, rather than increase, in the symbols created at each step is apparent: 8 at stage 1, 5 at stage 2, 3 at stage 3, 2 at stage 4, and 1 (the parse completion) at stage 5.

3. Complexity

The parallel parsing process described in Section 2 constructs all possible (partial) parses of σ , even mutually inconsistent ones; indeed, after n rounds of parallel rule applications, every possible sequence of $\leq n$ rule applications is represented by a path from $\$$ to $\$$ in the lattice graph. Thus if σ has a parse of length n , the path $\$ \sigma \$$ will occur in the graph, and indeed every sentential form in the parse will also occur as a path.

Of course, parallel parsing can potentially lead to a combinatorial explosion. Even if the graph itself does not become very large, the number of paths from $\$$ to $\$$ in the graph will grow, and these paths must (in principle) all be checked at each round for possible new rule applications. However, if the number and length of the rules are not too large, the amount of checking required cannot be very large. If the average (out) degree of a node of the lattice graph is d , the average number of strings of length k that start at a given node is d^k ; thus if all rule RHSs have length $\leq k$, we need only check d^k possibilities for each node, on the average. Moreover, it should be possible to use fast string matching techniques to reduce the amount of checking that is needed. The examples given in Section 2 suggest that at least for some grammars, the graph does

not grow rapidly, and multiple paths resulting from alternative choices at a given stage may recombine (i.e., lead to the same derived path) at a later stage.

A possible approach to reducing the combinatorial growth might be to apply only a subset of the rules that are applicable at a given stage, where the subset is chosen on heuristic grounds as being somehow "most likely" to lead to a parse - e.g., apply the rules whose RHSs are longest (so that they yield the shortest sentential forms), or the rules that lead to nonterminal symbols that are derivable from the start symbol in the fewest possible steps. However, it is easy to contrive grammars in which such heuristics would not lead to a successful parse. Another possibility might be to apply all possible rules at a given stage, but then allow the results to participate in a cooperation/competition process (e.g., rules that give rise to overlapping parts of σ compete, since they are mutually inconsistent; if a rule creates a symbol used by a later rule, the latter reinforces the former), and eliminate rule applications that have too much competition and not enough support. Here again, however, it is not hard to contrive examples in which this would eliminate rules that are necessary for a parse. The use of heuristics for rule selection, and cooperation/competition ("relaxation") for rule elimination, will therefore not be investigated here.

4. Implementation

The implementation of the parallel parsing process is straightforward, and does not require explicit construction of the lattice graph. Our method of implementation is based on the fact that for context-free grammars, any symbol appearing in a parse has a well-defined "scope" with respect to the original string σ , i.e., it arises from a specific substring of σ . Moreover, two symbols occur consecutively on a path in the lattice graph iff their scopes are consecutive substrings of σ . Based on these observations, given the string $\sigma \in x_0 \dots x_n$, we create for $0 \leq i \leq n$ a list of the symbols whose scopes begin in the i th position, and for each of these symbols we give the position at which its scope ends. Initially, L_i consists of x_i alone, with ending position i . To find the paths through the graph that begin with a particular symbol A (say on list L_i , with ending position j), we use the fact that the successors of A on all such paths are just the symbols on list L_{j+1} , and we repeat this process to find the subsequent symbols.

To apply a new rule, say $A \rightarrow B_1 B_2 \dots B_K$, we proceed as follows: Scan the lists for all occurrences of B_1 . For each occurrence, go to its successor list and check for the presence of B_2 ; for each of these, go to its successor list and check for B_3 ; and so on. If the rule has a short RHS, the number of possibilities to be checked should not be very large. If we find a match to the entire RHS, say with B_1 on list L_i and with B_K

having ending position j , we add an A to list L_i with ending position j . Note that two rule applications may lead to the same result, if there are two sequences $B_1, \dots, B_K$ in which the B_1 's are on the same list L_i and the B_K 's have the same ending position j ; in fact, two rules $A \rightarrow B_1 \dots B_K$ and $A \rightarrow C_1 \dots C_H$ may also lead to the same result if B_1, C_1 are both on L_i and B_K, C_H both end at j . After each round of rule applications, we should check each list L_i for duplicates (same symbol with same ending position) and eliminate them. If we want to maintain strict parallelism in applying the rules, the A 's that we find should be put on a separate set of lists L'_i , rather than on the current lists L_i ; when we have completed a round of rule applications, we append each L'_i to the corresponding L_i and check for duplicates.

Parallel rule application could be implemented by a multi-processor system as follows: Each processor maintains one of the lists L_i (or several, if there are more symbols in the initial string than there are processors). To apply the rule $A \rightarrow B_1 \dots B_K$, we broadcast it to all the processors. Any processor having B_1 's on its list sends messages to the processors responsible for the successor lists to check for B_2 's; and so on. If a sequence goes to completion, the processor that had the corresponding B_1 on its list L_i adds an A to L'_i . Note that the amount of time required for parallel application of $A \rightarrow B_1 \dots B_K$ is not simply proportional to K , since many messages may arrive at the same processor simultaneously and must then wait to be processed.

5. Generalization to arbitrary grammars

Our parallel rewriting process extends in principle to grammars that are not context-free, but there are some complications. Given a rule $\alpha \rightarrow \beta$, we can apply it by constructing a "bypass" (not necessarily a short-cut, since α may be longer than β) in which β follows the predecessor(s) of α and is followed by the successor(s) of α . If some of the symbols in α and β are the same, e.g., in the context-sensitive rule $\xi A \eta \rightarrow \xi a \eta$, it would be more economical to construct bypasses only for the new symbols, i.e., A follows the predecessors of α and is followed by its successors. Note, however, that this could give rise to paths from \$ to ¢ that could never occur as sentential forms.

To illustrate how the process might work in a non-context-free case, we give a context-sensitive example. The language $\{a^n b^n c^n \mid n \geq 1\}$ has the grammar

$S \rightarrow abc$	$S \rightarrow aTBc$
$T \rightarrow aTBC$	$T \rightarrow abC$
$CB \rightarrow BC$	$bB \rightarrow bb$ $Cc \rightarrow cc$

For the string aaabbbccc we have the following parallel parse:

\$	a	a	a	$\frac{b}{bB_1}$	$\frac{b}{b}$	$\frac{b}{b}$	$\frac{c}{C_1 c}$	$\frac{c}{c}$	$\frac{c}{c}$	¢
				$\frac{bB_1}{bB_2}$			$\frac{C_1 c}{C_2 B_2}$	$\frac{c}{C_2 c}$		
				$\frac{bB_2}{C_3}$	$\frac{C_2}{B_3}$	$\frac{B_2}{C_3}$	$\frac{C_2 c}{B_3}$			
				$\frac{C_3}{T_4}$	$\frac{B_3}{C_4}$	$\frac{C_3}{B_4}$				
				$\frac{T_4}{T_5}$						
				$\frac{T_5}{S_6}$						

Note that bB_2 and C_2c have the same positions as bB_1 and C_1c , but are not duplicates; B_2 has the other B_1 as a successor, and C_2 has the other C_1 as a predecessor. Similarly, C_4B_4 has the same position as C_2B_2 , but is not a duplicate (in fact it arose from B_3 and C_2 , which were produced only by rewriting C_2 and B_2); e.g., C_2 has b and B_1 as predecessors, but C_4 does not (fact, its sole predecessor is C_3 , which arose from rewriting B_1).

We see from these remarks that the simple implementation given in Section 4 for the context-free case does not generalize to the context-sensitive case. Rather, it becomes necessary to construct the lattice graph explicitly, with pointers from each symbol to its successors. The graph is likely to be bigger, and it becomes much more difficult to detect duplicate paths in the graph.

6. Concluding remarks

With the increasing availability of highly parallel hardware, a parallel approach to parsing may deserve serious consideration . The effectiveness of this approach depends on limiting the combinatorial growth of the parse graph, but in some cases this growth may not be excessively explosive.* If the strings to be parsed are not too long, parallel hardware is available, and processing time is a significant consideration, parallel parsing becomes an attractive alternative to conventional sequential methods.

*Experimental studies of the growth rate of the graph for various types of grammars are planned.

References

1. J. E. Hopcroft and J. D. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, Reading, MA, 1969, Ch. 11.
2. A. Rosenfeld, A. Y. Wu, and T. Dubitzki, Fast language acceptance by shrinking cellular automata, TR-898, Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park, MD, April 1980.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 81 -0623	2. GOVT ACCESSION NO. AD-A105028	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PARALLEL STRING PARSING USING LATTICE GRAPHS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-1036
7. AUTHOR(s) Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Center University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 31102F, 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB DC 20332		12. REPORT DATE May 1981
		13. NUMBER OF PAGES 16
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Formal languages Grammars Parsing Parallel processing		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Given a grammar G and a string σ , all possible parses of σ can be constructed by repeatedly applying the rules of G in parallel. This process creates a "lattice graph" in which any directed path from the least element to the greatest element is a sentential form that occurs in a (partial) parse of σ . Examples are given illustrating how, at least for some grammars, this process does not lead to a combinatorial explosion, and could thus be used to parse strings very rapidly if suitable parallel hardware were available.		